1.) Implement an algorithm to find the kth to last element of a singly linked list. Assume the size of the linked list is not initially known.

```
public LinkedListNode kthToLast(LinkedListNode head, int k) {
        LinkedListNode p1 = head;
        LinkedListNode p2 = head;

        for(int i = 0; i  <  k;  i++) {
                If(p1 == null) {
                        return null;
                }
                p1 = p1.next;
        }

        while(p1 != null) {
                p1 = p1.next;
                p2 = p2.next;
        }

        return p2;
}
```

2.)

```
public LinkedList314<E> getWithoutTarget(E tgt) {
    LinkedList314<E> result = new LinkedList314<>();
    Node<E> tempInThis = first;
    Node<E> tempInResult = null;
    while (tempInThis != null) {
        E val = tempInThis.data;
        if (!val.equals(tgt)) {
            // add to the result
            if (tempInResult == null) {
                // first element
                result.first = new Node<>(val);
                tempInResult = result.first;
            } else {
                // not first element
                tempInResult.next = new Node<>(val);
                tempInResult = tempInResult.next;
            }
        }
        tempInThis = tempInThis.next;
    }
    return result;
}
```

3.) Implement an algorithm to delete a node in the middle of a singly linked list, given ONLY access to that node (i.e any node in the list aside from the first or the last). If it cannot be deleted, return false.

```
public boolean deleteNode(LinkedList n) {
        if(n == null || n.next == null) {
                return false;
        }
        LinkedListNode next = n.next;
        n.data = next.data;
        n.next = next.next;
        return true;
}
```